Hardware Realization of an FPGA Processor - Operating System Call Offload and Experiences

Hindborg, Andreas Erik; Karlsson, Sven

Published in:
Proceedings of the 10th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES 2014)

Publication date:
2014

Hardware Realization of an FPGA Processor – Operating System Call Offload and Experiences

Andreas Erik Hindborg*,[1],
Sven Karlsson*,[1]

* *DTU Compute, Technical University of Denmark Richard Petersens Plads, Building 324, 2800, Kgs. Lyngby, Denmark*

ABSTRACT

Field-programmable gate arrays, FPGAs, are attractive implementation platforms for low-volume signal and image processing applications.

The parallel structure of FPGAs allows for an efficient implementation of parallel algorithms. Sequential algorithms, on the other hand, often perform better on a microprocessor. It is therefore convenient for many applications to employ a synthesizable microprocessor to execute sequential tasks and custom hardware structures to accelerate parallel sections of an algorithm. In this paper, we discuss the hardware realization of Tinuso-I, a small synthesizable processor core that can be integrated in many signal and data processing platforms on FPGAs. We also show how we allow the processor to use operating system services. For a set of SPLASH-2 and SPEC2006 benchmarks we show an speedup of up to 64% over a similar Xilinx MicroBlaze implementation while using 27% to 35% fewer hardware resources.

1 Introduction

The ever increasing cost of developing a custom designed application specific integrated circuit, *ASIC*, has long since passed the point of feasibility for low volume embedded systems to include such custom components. Instead, designers look to FPGA devices for low volume markets, especially for signal and image processing. The performance and unit price of custom FPGA deigns are orders of magnitude lower and higher, respectively, than for custom designed ASICs. However, the attainable performance may be orders of magnitude higher than a solution that uses generalized components.

Not all algorithms benefit equally from implementations on FPGAs. While it can be very efficient to implement parallel algorithms on FPGA devices, sequential algorithms are often better implemented on microprocessors. Therefore, signal processing applications often include microprocessors in the FPGA fabric. In this paper, we present our experiences in realizing a custom processor core that targets FPGA implementation. While we previously have used simulation to verify designs, in this paper we discuss how we realized the processor core, the *Tinuso-I*, on the Xilinx Zynq SoC platform.

[1]E-mail: {ahin,svea}@dtu.dk

We propose and implement a method for offloading operating system services in an embedded system. We present specially designed interface logic and run-time libraries that enable operating system calls to be made from a Tinuso-I [SMK+12] instance running in the FPGA fabric.

We evaluate the system by executing selected SPEC 2006 and SPLASH-2 benchmarks. We demonstrate an speedup up to 64% over a Xilinx MicroBlaze based baseline system.

Several software layers and components are needed to execute real applications. Applications are commonly developed assuming a POSIX compliant operating system. However, running a full operating system on small embedded systems is often unfeasible.

We will use relatively large benchmarks for illustration. Such benchmarks are normally used to evaluate application processors such as mobile phone processors, tablet processors, desktop processors and server processors. Examples are the programs in the SPEC2006 and SPLASH2 suites.

To run said benchmark applications, certain POSIX operating system services must be available. For the SPEC2006 programs, it is enough to provide the **open**, **close**, **read**, **write**, **fstat** and **fseek** POSIX system calls. The SPLASH2 applications also require the **gettimeofday** system call which is used to provide detailed timing metrics of the benchmark execution.

We intercept file system service requests by linking the benchmark applications with a custom run-time library. When an application requests a file system service, such as invoking the **open** system call, the run-time library sends the request to a PC via a communication link. The response is received by the run-time and relayed back to the requesting program. With this approach it is possible to provide file system access to a processor core by only implementing the following: *a*) A minimal run-time library that intercepts file system service requests from programs running on the Tinuso-I core in FPGA fabric. *b*) Communication link support on the PC that services the file system requests and on the soft core. *c*) A service on the PC that responds to the file system service requests.

We implement the proposed offloading method for a Tinuso-I core synthesized to the FPGA fabric of a Xilinx XC7Z020-CLG484-1 Zynq device. The silicon device is part of an AvNet ZedBoard development kit. The Zynq device is a complex unit with many components, including two ARM Cortex A9 cores, an 1G Ethernet MAC and an FPGA fabric area. We use TCP/IP over the 1G Ethernet MAC to provide the communication link between Tinuso-I and the PC.

We compile applications for the Tinuso-I using an embedded binutils/GCC/Newlib tool chain. A Standard C library API are provided by the Newlib C library. Newlib services file system requests by calling user supplied methods implementing the service. We compare the system to a similar MicroBlaze system. The system architectures are depicted in figure 1.

2 Results

The Tinuso system utilizes about 27% fewer registers and 35% fewer LUTS than a similar MicroBlaze based system.

The execution time for the benchmarks are depicted in figure 2a. The figure shows the execution time for the benchmarks when executed on a 168 MHz Tinuso-I system and a 115 MHz MicroBlaze system, normalized to the latter. The results show that the Tinuso-I system achieves a speedup of up to 64%.

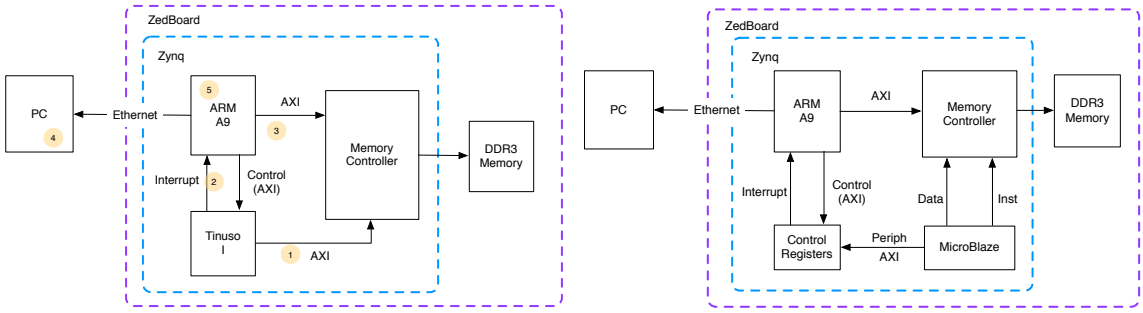

Figure 1: (a) Test system architecture overview. A single Tinuso-I core is instantiated in the FPGA fabric of a Xilinx Zynq device. File system access is provided by a PC connected by Ethernet. (b) Test sytem architecture for the MicroBlaze baseline system.

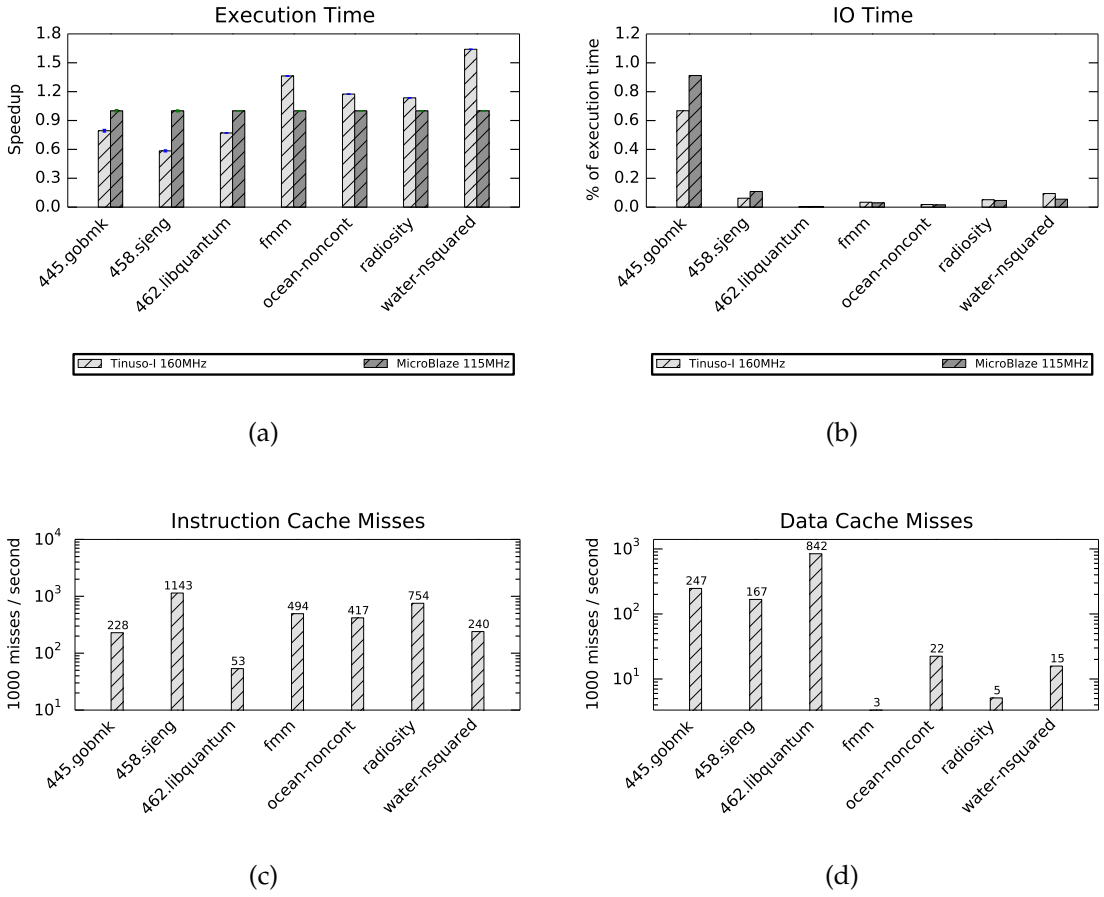

Figure 2: (a) Execution time of the benchmarks normalized to the execution time on the MicroBlaze system. (b) Average fraction of execution time spent in the IO system. (c) Number of instruction cache misses normalized to execution time. (d) Number of data cache misses normalized to execution time.

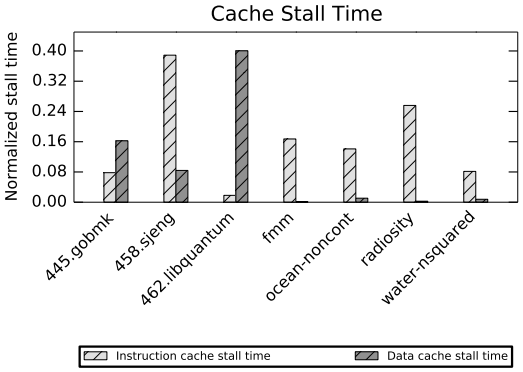

Figure 3: Time spent waiting for cache.

Figure 2b shows the average time spent in the IO system. The IO time is just a tiny fraction of the entire execution time, no larger than 1% and in most cases less than 0.2%.

Figure 2c presents the number of misses in the instruction cache divided by the execution time of the program. Figure 2d shows the same metric, but for the data cache. For the three SPEC benchmarks where Tinuso-I is slower than MicroBlaze, the number of data cache misses is significantly higher. Figure 3 shows the fraction of the execution time that is spent waiting for the caches by the Tinuso-I system. For **458.sjeng** and **462.libquantum**, execution time is dominated by cache stalls.

3 Conclusion

We have shown how we used the Xilinx Zynq SoC to realize the Tinuso-I processor core and to perform a hardware bringup. We have proposed a method for offloading operating system services using the ARM host of the Xilinx Zynq SoC. This proposed system for offloading operating system services is highly relevant for prototyping and simulating signal and data processing applications. We have demonstrated our method by using it to evaluate both Tinuso-I and Xilinx MicroBlaze. We evaluate the system by executing a set of SPEC 2006 and SPLASH-2 benchmarks. We demonstrate a speedup of up to 64% over a similar Xilinx MicroBlaze baseline system. On average Tinuso-I performs 6% better than MicroBlaze while utilizing 27% fewer LUTs and 35% fewer registers. Tinuso-I is highly configurable and the simple architecture allows for an easy extension it with dedicated hardware blocks. Tinuso therefore is an attractive platform for a broad range of embedded system signal and data processing applications.

References

[SMK+12] Pascal Schleuniger, Sally A. McKee, Sven Karlsson, Andreas Herkersdorf, Kay Romer, and Uwe Brinkschulte. Design principles for synthesizable processor cores. In *Architecture of Computing Systems–ARCS 2012*, pages 111–122. Springer, February 2012.